Fichas iOS

autentia

Como en anteriores ocasiones, Autentia se embarca de nuevo en la publicación de contenidos didácticos en forma de fichas. Nuestra experiencia anterior con las fichas Agile ha sido y continúa siendo un éxito de aceptación en la comunidad.

Esto nos ha impulsado a realizar una nueva publicación con una **recopilación de buenas prácticas, patrones, principios y tecnologías para el desarrollo de software** que han ido componiendo la serie de publicaciones sobre guías técnicas que usamos en el propio Onboarding de Autentia.

Las guías para directivos y técnicos que Autentia lleva publicando desde finales de la primavera de 2020, se han concebido ya con la incorporación de fichas que aunque integradas totalmente en cada uno de los documentos de la serie, permiten también agruparlas por temas y manejarlas de manera independiente haciendo de ellas un activo muy valioso que se puede consultar de una manera cómoda y rápida.

Estas fichas exponen conceptos, herramientas y procesos muy utilizados tanto a nivel de negocio como a nivel tecnológico pero todas con un interés y un valor apropiado que estamos seguros que la comunidad volverá a valorar.

Attribution-ShareAlike 4.0 International
(CC BY-SA 4.0)

Esta obra está licenciada bajo licencia Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)

Definición

Es un framework de Apple para el desarrollo de **Machine Learning** en apps nativas. Con posibilidad de entrenamiento en el dispositivo. Xcode viene acompañado de la herramienta **CreateML** para generar los modelos y entrenarlos sin una sola línea de código.

VENTAJAS DE NEGOCIO

Conocimiento profundo del cliente, por ejemplo:

- Categorizar los clientes en función de su comportamiento para una correcta segmentación.
- Conocer y predecir la Navegación dentro de nuestra app para saber dónde hacer foco.
- Conocer y predecir las búsquedas del cliente concreto para ofrecerle de manera activa los productos demandados.

Recomendar los productos correctos:

- Conocer patrones ocultos en el comportamiento de un cliente para ofertar productos que le han interesado a otros clientes de comportamiento similar.
- Hacer estrategias de ventas y marketing basadas en patrones de comportamiento general o individual.

VENTAJAS DE DESARROLLO

- **Soporte desde iOS 11** y entrenamiento “on-device” desde iOS 13.
- **Rápida implementación** con una API simple y clara.
- CoreML es un framework **nativo de Apple**, lo que nos garantiza su continuidad y calidad.
- Mediante la herramienta de *Xcode CreateML* **podemos crear de forma rápida y sin código nuestros modelos y entrenarlos.**
- Mediante *coreMLtools* **podemos transformar modelos hechos en otras plataformas al formato de Apple**.

```
extensión MyModel {
    func predict(_ value: MLFeatureValue) -> String {
    guard let pixelBuffer = value.imageBufferValue,
        let prediction = try? prediction(image:
pixelBuffer).classLabel else {
        return "unknown"
        }
    return prediction
    }
}
```

Definición

Es un framework de Apple para la construcción de **interfaces visuales** de forma declarativa, reduciendo la complejidad del código y optimizado para aprovechar las funciones nativas del sistema. Disponible desde macOS Catalina 10.15, iOS 13, iPadOS 13, tvOS 13 y watchOS 6.

VENTAJAS

- Sintaxis declarativa y programación reactiva.
- Diseño bidireccional con las previews: añades un objeto a la interfaz y se actualiza el código y añades código y se actualiza la interfaz.
- Con las Previews puedes ejecutar tu app en un dispositivo Apple conectado y hacer cambios en tiempo real.
- Soportado en todas las plataformas (iOS, iPadOS, macOS, watchOS y tvOS).
- Animaciones complejas con poca codificación.
- Favorece la implementación de una arquitectura limpia con MVVM o VIPER.
- Se puede mezclar con UIKit sin problema.
- Prescinde de Interface Builder y Autolayout.
- Es el futuro de la construcción de interfaces, tanto Apple como la comunidad apuesta fuertemente por esta tecnología.

EJEMPLO

```
struct MasterView: View {
    @State private var showPopover: Bool = false

    var body:some View {
        VStack {
            Button("Show popover") {
                self.showPopover = true
            }.popover( isPresented: self.$showPopover) {
                Text("Popover")
                        .bold()
                        .underline()
                        .foregroundColor(.red)
                }
            }
      }
}
```

¿Qué es?

Es un lenguaje de programación desarrollado por Apple para ser utilizado en sus plataformas con la intención de ser una alternativa a Objective-C. Inicialmente propietario pero liberado como código abierto en su versión 2.2.

¿PARA QUÉ SE USA?

Inicialmente, cuando fue presentado por Apple, su propósito era ser una alternativa a Objective-C para el desarrollo de aplicaciones nativas para las plataformas Apple (i**OS, iPadOS, macOS, watchOS y tvOS**).

Con la versión 2.2 y tras ser liberado como código abierto, ha sido portado a Linux y Windows y han surgido varios proyectos open source que hacen uso del mismo. Sobre el papel, podría ser utilizado para abordar cualquier desarrollo.

En la actualidad, también puede ser utilizado para el desarrollo de backend, por ejemplo con **VAPOR**. También fue adoptado por Google para usarlo con **TensorFlow** como alternativa o complemento a Python.

Recientemente se ha anunciado un proyecto para poder utilizar Swift también con AWS lambda.

CARACTERÍSTICAS

- Interoperable con Objective-C (y por extensión, con C y C++ mediante un wrapper). Posteriormente añadida interoperatividad con Python.
- Lenguaje fuertemente tipado.
- Compilado.
- Orientado a protocolos.
- Inferencia de tipos.
- Distinción entre tipos de datos opcionales y no opcionales para prevenir los problemas relacionados con la nulabilidad.
- Cuenta con características de programación funcional.
- Closures (conocidos como lambdas en otros lenguajes) y funciones first-class.
- Multiplataforma con soporte para macOS, Linux y Windows.

Definición

SPM es un **gestor de dependencias** basado en paquetes. Tiene soporte para aplicaciones en lado servidor y aplicaciones de front nativas de Apple. Se puede crear, ejecutar y desplegar por línea de comandos desde el lanzamiento de Swift 3.0, o a través de Xcode desde Swift 5.0.

¿QUÉ ES UN PAQUETE?

Un paquete es una colección de archivos de código en Swift que conforman una librería, también contiene un fichero llamado **Package.swift** que es el manifiesto en el que se declara la configuración del propio paquete. La configuración básica de un paquete contiene:

- El nombre del paquete.
- Las plataformas soportadas.
- Los productos que genera el paquete (las librerías).
- Las dependencias (puede contener dependencias con otros paquetes).
- Los targets que contiene (Módulos).

¿PARA QUÉ SIRVE?

Cuando necesitamos importar librerías propias o de terceros a nuestro proyecto, vamos a enfrentarnos al problema de ver cómo gestionar las actualizaciones y dependencias que éstas tienen.

A día de hoy, en el mundo del desarrollo iOS, existen soluciones como Cocoapods o Carthage que dan más problemas que soluciones. Con **SPM** vamos a tener las siguientes ventajas:

- Integración total en Xcode.
- Funciona en Linux.
- Linkado automático.
- Gestiona automáticamente las dependencias de los paquetes.
- De código abierto.

¿CÓMO LO USO?

Puedes crear un paquete por línea de comandos:

$ swift package init

Con de Xcode:

File -> New -> Swift Package.

Una vez creado se sube a un repositorio público de Git.

Para importar tu paquete desde Xcode:

File -> Swift Packages -> Add Package Dependency

También puedes importar tu paquete como dependencia de otro paquete en el archivo **Package.swift**.

Definición

Es un proyecto de Apple para transformar las aplicaciones desarrolladas para las plataformas iOS/iPadOS en aplicaciones para Mac. Es importante tener en cuenta que solo las apps con versión para iPad van a poder adaptarse.

¿CÓMO?

La transformación se hace **traduciendo componentes** de UIKit (iOS) en AppKit (macOS).

Hay que tener en cuenta que hay frameworks de iOS que no tienen su contrapartida en Mac, por lo que si usas ARKit, HealthKit, la cámara o el giroscopio en tu aplicación, tendrás que tenerlo en cuenta en tu código:

```
#if !targetEnvironment(macCatalyst)
// Code to exclude from Mac.
#endif
```

¿CUÁNDO?

No todas las apps van a tener sentido en macOS, si tu aplicación se basa en la navegación GPS, o entrenar en un gimnasio, Apple no la va a aprobar para su publicación en el Store.

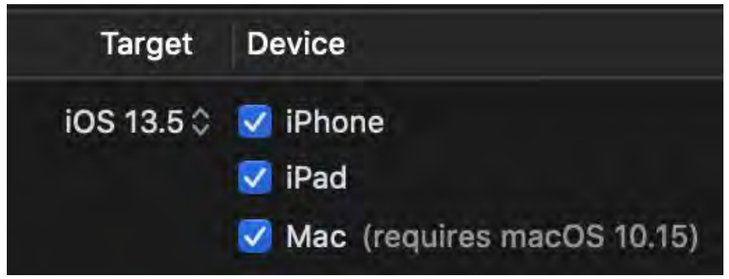
Cuando tu aplicación sea idónea para la transformación en una versión para Mac, vas a tener que implementar una serie de características que igual no tienes implementadas en iOS:

- Soporte para multitarea.
- Drag and drop.
- Atajos de teclado.

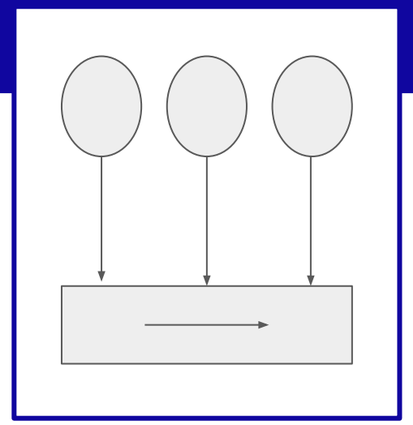
¿DÓNDE?

Para hacer tu app compatible con Mac basta con tener instalado macOS 10.15 y Xcode 11.

En la pestaña **General** de tu proyecto y en el apartado **Deployment Info** activar el checkbox de Mac:



Una vez activado podrás correr la app en el target de macOS desde tu Scheme.



Definición

Es un framework de Apple que nos proporciona una API declarativa para el proceso de valores a lo largo del tiempo (programación reactiva).

CARACTERÍSTICAS PRINCIPALES

Los conceptos básicos que hemos de entender en **Combine** son dos:

- **Publisher:** el publicador es un tipo de objeto **observable** que emite valores a lo largo del tiempo. Un publicador tiene operadores para actuar sobre valores recibidos de otros publicadores y volver a publicarlos.
- **Subscriber:** el suscriptor es un tipo de objeto que **escucha los valores** generados por los publicadores. Los publicadores solo emiten valores cuando el suscriptor los solicita.

Se puede combinar la salida de diferentes publicadores y coordinar si interacción.

```
let publisher = PassthroughSubject<Int, Never>()

let subscription = publisher
  .sink { value in
          print("Subcription received integer: \(value)")
}
```

COMBINE + FOUNDATION

Con el lanzamiento de Combine, Apple ha añadido muchas extensiones en su framework Foundation para añadir las capacidades y **ventajas de Combine** en muchas de sus clases. Por ejemplo:

- URLSession.
- Timer.
- NotificationCenter.
- Sequence.

Definición

Es un framework de Apple para crear experiencias de **realidad aumentada.** Trabajando junto con **RealityKit** y la herramienta **Reality Composer**, dispondremos de todo lo necesario para desarrollar nuestras apps de AR.

LO QUE HAY QUE SABER

- **ARKit** es un framework para obtener información del mundo real. A partir de su versión 3, junto con RealityKit soporta oclusión automática en tiempo real, seguimiento facial de hasta 3 personas simultáneas y captura de movimiento en tiempo real.

- **RealityKit** es un framework que ofrece una API para renderizado en 3D, utiliza la información proporcionada por ARKit para integrar fácilmente objetos virtuales en entornos del mundo real, con escalado automático, renderizado fotorealista, física y animaciones.

- **Reality Composer** es una herramienta de Xcode para crear y editar escenas de AR sin código, permite importar archivos USDZ y animarlos, así como desencadenar eventos con distintos tipos de inputs.

LOS COMPAÑEROS PRINCIPALES

ARKit tiene 4 compañeros para el renderizado de gráficos:

- RealityKit.
- SceneKit.
- SpriteKit.
- Metal.

VENTAJAS

- Está **completamente integrado en el ecosistema de Apple** y se puede usar con otros frameworks de Apple como SceneKit o SpriteKit, así como en las apps Mac Catalyst.

- Es **bastante sencillo de usar** ya que tiene muchas cosas hechas e integradas por Apple como el rastreo de la cara, efectos especiales, etc.

Definición

Es un **framework de Apple para dibujar formas, partículas, texto, imágenes y vídeo en dos dimensiones.** Aprovecha Metal para lograr alto rendimiento, al tiempo que ofrece una interfaz de programación simple para facilitar la creación de juegos y otras aplicaciones intensivas en gráficos. SpriteKit es compatible con iOS, macOS, tvOS y watchOS, y se integra bien con los frameworks como GameplayKit y SceneKit.

BLOQUES PRINCIPALES

SKView: una vista en la que se presentan *SKScenes*.

SKScene: una escena 2D que se presenta en un *SKView* y contiene uno o más *SKSpriteNodes*.

SKSpriteNode: una imagen 2D individual que puede ser animada alrededor de la escena.

BLOQUES DE CONSTRUCCIÓN

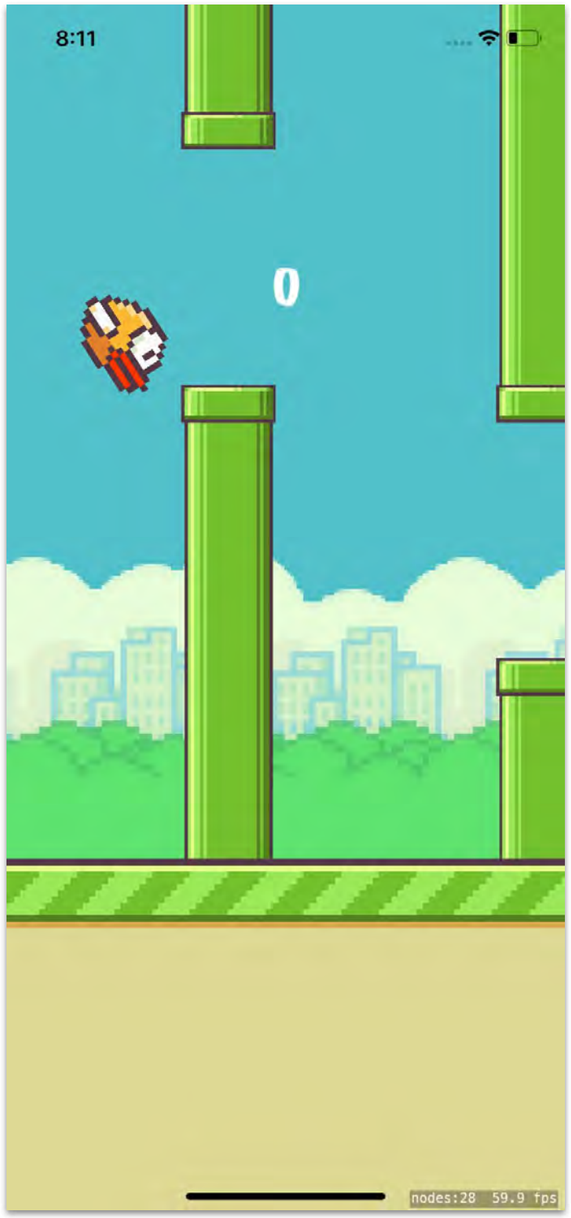
SKNode: un nodo más general que se puede usar en una escena para agrupar otros nodos para un comportamiento más complejo.

SKAction: acciones individuales o grupos de acciones que se aplican a *SKNodes* para implementar animaciones y otros efectos.

SKPhysicsBody: permite que la física se aplique a *SKNodes* para permitir que se comporten de una manera realista, incluida la caída por gravedad, rebotar entre sí y seguir trayectorias balísticas.

VENTAJAS

- Está integrado en el ecosistema de Apple.
- Es bastante fácil de utilizar.
- Es rápido gracias a Metal.
- Es gratuito.
- Puedes utilizarlo en todas las plataformas de Apple.
- Tiene el motor de física integrado.



Definición

Es un **framework de Apple para construir las aplicaciones de gráficos 3D de alto nivel** que ayuda a crear escenas y efectos animados en 3D. Incorpora un motor de física, un generador de partículas y formas fáciles de escribir las acciones de los objetos 3D para que puedan describir su escena en términos de su contenido (geometría, materiales, luces y cámaras) y luego animarla describiendo los cambios en esos objetos.

CARACTERÍSTICAS PRINCIPALES

El framework de SceneKit fue lanzado por primera vez por Apple junto con OS X 10.8 Mountain Lion y más tarde estuvo disponible en iOS con el lanzamiento de iOS 8.

El **propósito de éste framework es permitir a desarrolladores, integrar fácilmente gráficos 3D** en juegos y aplicaciones sin las complejidades de APIs para gráficos, tales como OpenGL y Metal. Todos los recursos, representados por nodos, son arreglados en un árbol jerárquico llamado **scene graph**.

Éste árbol funciona muy similar a un **view hierarchy** regular en UIKit. SceneKit te permite simplemente proporcionar una descripción de los recursos que deseas en su escena, con el propio framework que maneja todo el código de representación OpenGL para ti.

BLOQUES PRINCIPALES

SCNView: una vista en la que se presentan *SCNScenes*.

SCNScene: una escena 3D que se presenta en un *SCNView*

SCNVector: un vector de tres componentes que representa la posición de un nodo.

SCNNode: un elemento estructural de un gráfico de escena que representa una posición y se transforma en un espacio de coordenadas 3D, al que puede adjuntar geometría, luces, cámaras u otro contenido visualizable.

Definición

Es un framework de Apple que gestiona las peticiones de los usuarios para los servicios de las aplicaciones que se originan en Siri o Maps.

CARACTERÍSTICAS PRINCIPALES

SiriKit abarca los frameworks de *Intents* e *Intents UI*, que utiliza para implementar extensiones de aplicaciones que integran sus servicios con *Siri* y *Maps*. *SiriKit* admite dos tipos de extensiones de aplicación:

- **Una extensión de aplicación *Intents*** recibe solicitudes de los usuarios de *SiriKit* y las convierte en acciones específicas de la aplicación. Por ejemplo, el usuario puede pedirle a Siri que envíe un mensaje, reserve un viaje o comience un entrenamiento con su aplicación.

- **Una extensión de la aplicación *Intents UI*** muestra la marca u otro contenido personalizado en la interfaz Siri o Maps, después de que la extensión de la aplicación *Intents* cumple una solicitud del usuario. La creación de esta extensión es opcional.

BLOQUES PRINCIPALES

SiriKit define los tipos de solicitudes, conocidos como **Intents**, que los usuarios pueden realizar.

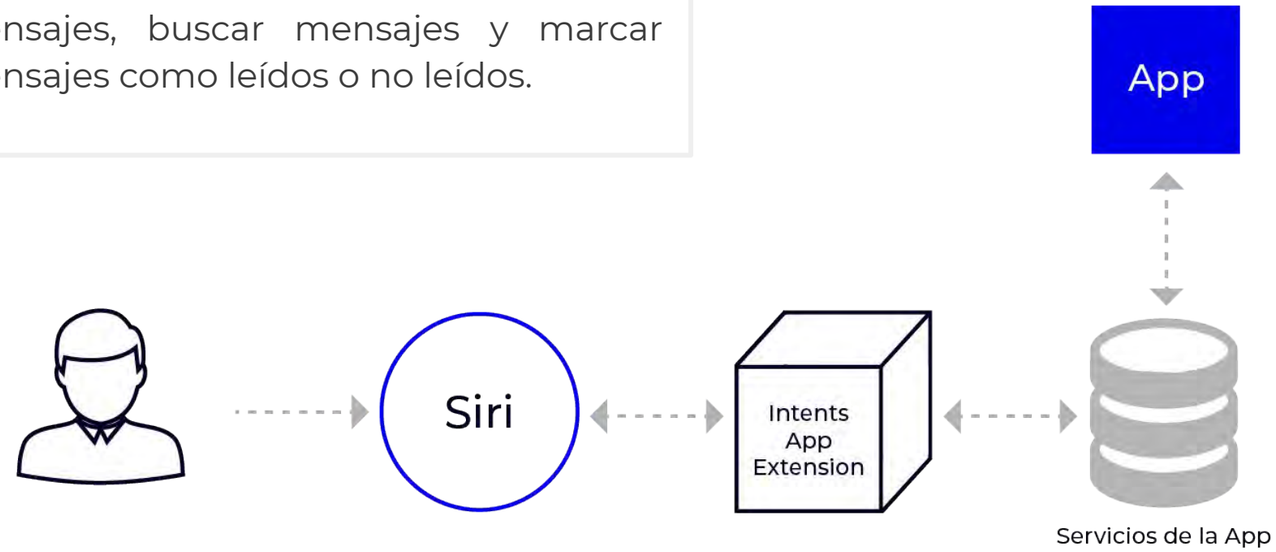
Los **dominios** (domains group) agrupan las intenciones relacionadas para dejar claro qué intenciones puede soportar la aplicación.

Por ejemplo, el dominio de mensajes puede tener el intento de enviar mensajes, buscar mensajes y marcar mensajes como leídos o no leídos.

RESTRICCIONES

- Hay que pedir el permiso al usuario.

- El sistema provee unos dominios predefinidos (envío de mensajes, establecer llamadas de voz sobre IP, pagos, etc.), en caso de que no se ajuste a lo deseado es necesario implementar un Intent personalizado.



Definición

Es un **framework de Apple para persistir o almacenar en caché los datos en un solo dispositivo.** Abstrae los detalles de mapear los objetos a un store (almacén), lo que facilita guardar datos de Swift y Objective-C sin administrar una base de datos directamente.

CARACTERÍSTICAS PRINCIPALES

Usa Core Data para guardar los datos permanentes de la aplicación para uso fuera de línea, almacenar datos temporales en caché y agregar funcionalidad de deshacer a su aplicación en un solo dispositivo.

- **Persistencia.**
- **Abstracción.**
- **Deshacer** y **rehacer** cambios individuales o por lotes.
- **Tareas de datos** en segundo plano (background tasks).
- **Versionado** y **Migración.**
- **Sincronización** de los datos con las vistas .

BLOQUES PRINCIPALES

- **Persistent store coordinator:** es un wrapper de nuestra base de datos, administra las conexiones (lecturas y escritura), lo podemos visualizar como un apuntador a nuestra base de datos.

- **NSManagedObjectModel:** describe el schema de nuestra base de datos, es decir, las tablas y relaciones.

- **NSManagedObjectContext:** nos permite crear, solicitar o actualizar objetos de nuestra base de datos.

- **Persistent container:** encapsula todos los componentes anteriores.

VENTAJAS

- Está integrado en el ecosistema de Apple

- El agrupar, filtrar u organizar nuestros datos.

- Evitar escribir SQL y lograr queries complejos mediante el uso de objetos Fetch Requests y Predicates.

- Reducir el impacto a nivel memoria (memory footprint) mediante el uso de Faulting.

Definición

Es un **framework de Apple para para incrustar los mapas de Apple directamente en tus propias vistas.** Proporciona la interfaz para añadir tus propias capas, anotaciones o texto encima del mapa.

CARACTERÍSTICAS PRINCIPALES

Usa el framework **MapKit para incrustar mapas directamente en tus vistas**. Puedes agregar anotaciones y superposiciones al mapa, los puntos de interés, etc. También proporciona el mecanismo para buscar los puntos de interés.

Si tu aplicación ofrece indicaciones de tránsito, **puedes hacer que tus indicaciones estén disponibles en la app Mapas**. Además, puedes usar la app Mapas para complementar las instrucciones que proporcionas en tu aplicación.

Por ejemplo, si tu aplicación solo proporciona indicaciones para viajar en metro, puedes usar Maps para proporcionar indicaciones a pie desde y hacia las estaciones de metro.

BLOQUES PRINCIPALES

- **MKMapView:** una vista de mapa incrustable, similar a la proporcionada por la aplicación Mapas.
- **MKMapItem:** un punto de interés en el mapa.
- **MKAnnotationView:** la representación visual de uno de tus objetos de anotación.
- **MKDirections:** un objeto de utilidad que calcula direcciones e información de tiempo de viaje en función de la información de ruta que proporciones.

VENTAJAS

- Está integrado en el ecosistema de Apple.
- Es bastante fácil de utilizar.

RESTRICCIONES

- La cobertura de los mapas aún es mejorable en muchos sitios.

Definición

Es un **framework de Apple para detectar etiquetas NFC**, leer la información guardada en ellas en el formato NDEF y modificarla.

CARACTERÍSTICAS PRINCIPALES

El framework Core NFC sirve para leer etiquetas para dar a los usuarios más información sobre su entorno físico y los objetos del mundo real.

Con Core NFC, **puedes leer las etiquetas de comunicación de campo cercano (NFC) de los tipos 1 a 5 que contienen datos en el formato de intercambio de datos NFC** (NDEF) y conectar con etiquetas ISO7816, ISO15693, FeliCa y MIFARE.

Por ejemplo, la aplicación podría proporcionar a los usuarios información sobre los productos presentados en una tienda o los cuadros en un museo.

BLOQUES PRINCIPALES

- **NFCReaderSession:** una clase de sesión de lectura para detectar etiquetas de formato de intercambio de datos NFC (NDEF) con NFCNDEFReaderSession y conectar con el resto de tags con NFCTagReaderSession.
- **NFCNDEFReaderSessionDelegate:** un protocolo de delegado que sirve para leer los datos NDEF de una etiqueta NFC.
- **NFCTagReaderSessionDelegate:** un protocolo de delegado que sirve para gestionar la conexión a un tag.
- **NFCNDEFPayload:** el payload (carga útil) que contiene los datos, el nombre de tipo de formato de NDEF de un mensaje, etc.
- **NFCTag:** representa el tag NFC al que se ha conectado.

RESTRICCIONES

- Core NFC **no está disponible para su uso en extensiones de aplicaciones** y requiere de un dispositivo que admita la comunicación de NFC.
- Core NFC **no admite ID de aplicaciones relacionadas con el pago.**
- **El uso de Core NFC es restringido** y es apto solo para trabajar con las etiquetas NFC y no se permite la comunicación entre otros dispositivos que soportan NFC.

Definición

Es un framework de Apple que proporciona la infraestructura para crear aplicaciones watchOS.

CARACTERÍSTICAS PRINCIPALES

Con la ayuda de framework **WatchKit** podemos crear simples y sencillas aplicaciones para WatchOS.

Se construyen con los elementos predeterminados y dentro de la extensión delegado (*extension delegate*).

A pesar de su simplicidad tienen acceso a las tareas en segundo plano, intenciones de Siri, sesiones de entrenamiento, etc.

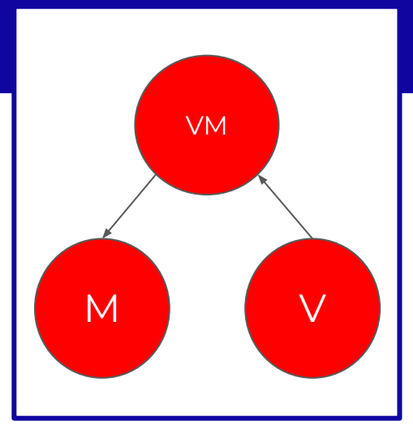
Los usos **más adecuados para apps WatchOS son mediciones de todo tipo**, las apps con solo una función principal (como Shazam), rastreadores de ejercicios, mostrar la información en el formato compacto, etc.

BLOQUES PRINCIPALES

- **WKExtension**: una clase de la extensión que gestiona los comportamientos de los controladores de interfaz de tu aplicación.
- **WKInterfaceController:** una clase que proporciona la infraestructura para administrar la interfaz en una aplicación watchOS (al igual que UIViewController en UIKit app).
- **WKExtensionDelegate:** el protocolo que contiene una colección de métodos que administran el comportamiento a nivel de aplicación de una extensión WatchKit.

RESTRICCIONES

- **Se necesita la app-compañero** (a partir de WatchOS 6 se puede crear las apps independientes).
- **Las animaciones están restringidas**. Puedes usar solo un conjunto de imágenes para crear una especie de GIF animado.
- **Las interfaces se crean con los elementos simples**, poco flexibles (una lista, un botón, un label, etc.).
- **Las notificaciones pueden ser de dos tipos:** *Long* and *Short*.
- **Los gestos están predeterminados**, no se soporta multitouch (multitáctil).
- **No puedes crear tus propios componentes personalizados**.



Definición

El patrón de arquitectura MVVM **consiste en separar nuestra aplicación en tres capas**: la lógica de negocio, la interfaz gráfica y la lógica de presentación.

CARACTERÍSTICAS PRINCIPALES

El **principal objetivo de este patrón es sacar el estado de la vista y la lógica de presentación de la vista**. De este modo, la vista solo contiene los elementos visuales.

El ViewModel **representa el estado de la vista y maneja los componentes de la vista y sus estados a través de "binding"** (atadura). El binding se puede hacer como un cierre (closure), a través del mecanismo Key-Value Observing o con la ayuda de las librerías de terceros, así como RxSwift/ReactiveCocoa, etc.

SwiftUI ya tiene los mecanismos integrados de "binding" como los property wrappers @State o @Binding.

BLOQUES PRINCIPALES

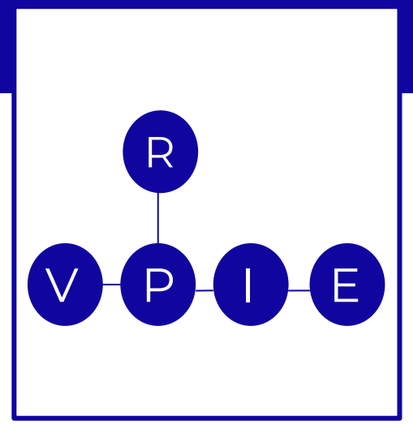
- **Model:** representa los conceptos y las entidades de negocio, también la lógica de negocio.
- **ViewModel:** contiene la lógica de presentación y el estado de la vista. Recupera los datos del negocio desde el Modelo, aplicando la lógica de presentación y guardando el estado de la vista. El estado se puede hacer global (a la Redux) o guardando cada viewModel su propio estado.
- **View:** pinta la interfaz gráfica y pasa las interacciones de usuario a ViewModel. Recupera los datos preparados desde ViewModel a través de binding.

PROBLEMAS

- Con el crecimiento de la lógica de la app el ViewModel puede convertirse en un mega-objeto poco sostenible.
- El problema de la gestión del estado de la vista es bastante complejo.

VENTAJAS

- Es un patrón bastante "sencillo" para implementar sobre todo para las apps simples.
- Alta posibilidad de reutilización de los ViewModels para diferentes Views.



Definición

Viper **es un patrón arquitectónico que implementa los principios de *Clean Architecture***, aplicándolos al desarrollo de aplicaciones iOS. Consiste en separar la lógica de la aplicación en cinco capas: View, Interactor, Presenter, Entity y Router.

CARACTERÍSTICAS PRINCIPALES

La implementación del patrón Viper en el desarrollo de aplicaciones iOS **consiste en dividir la lógica de cada módulo (pantalla) de la aplicación en cinco capas principales** con las diferentes responsabilidades.

De esto modo, **aislamos las dependencias y podemos testear más fácilmente** tanto las interacciones entre capas como las capas propias.

Las **interacciones entre capas se ejecutan a través de los protocolos** para poder sustituir los objetos reales con los mocks en los tests unitarios.

BLOQUES PRINCIPALES

- **View:** pinta lo que le manda el Presenter y pasa las entradas de usuario al Presenter. Es la capa más sencilla de todas.
- **Interactor:** contiene la lógica de negocio.
- **Presenter:** contiene la lógica de presentación y procesa las entradas de usuario.
- **Entity:** contiene los modelos básicos (entities) de negocio que se usan por Interactor.
- **Router:** contiene la lógica de navegación.

DESVENTAJAS

- Existe cierta “sobrecarga” en crear cinco clases diferentes con los diferentes protocolos por cada módulo.
- Es un patrón complejo que requiere cierto aprendizaje y costumbre de usar.

VENTAJAS

- Cada capa es reutilizable y testeable.
- Con el crecimiento, la aplicación sigue estando estructurada y testeable a pesar de su tamaño.

Definición

La accesibilidad en las aplicaciones móviles significa hacerlas manejables y cómodas para toda la población, independientemente de sus capacidades técnicas o físicas.

LAS HERRAMIENTAS PARA DESARROLLADORES

- **Accessibility Inspector:** es una herramienta que nos proporciona la información sobre la accesibilidad de cada elemento de nuestra aplicación. Además, puede hacer la auditoría de toda nuestra aplicación en términos de accesibilidad.

- **Accessibility Labels:** cada vista UIView tiene la propiedad accesibilityLabel que facilita la información sobre la vista al lector de pantalla VoiceOver.

- **Accessibility Traits:** para los elementos customizados es importante poner un rasgo(trait) para transmitir el sentido del elemento.

- **Accessibility Notifications:** puedes crear las notificaciones que avisan al usuario sobre los cambios en la interfaz, scroll o cualquier anuncio en la voz alta.

- **DynamicTypes:** escala los textos en la aplicación según los ajustes del móvil.

LAS HERRAMIENTAS PRINCIPALES PARA USUARIO

- **VoiceOver:** el lector de pantalla que pronuncia en voz alta todo lo que sucede en la pantalla y se manipula con los gestos.

- **Zoom:** escala el contenido de la pantalla.

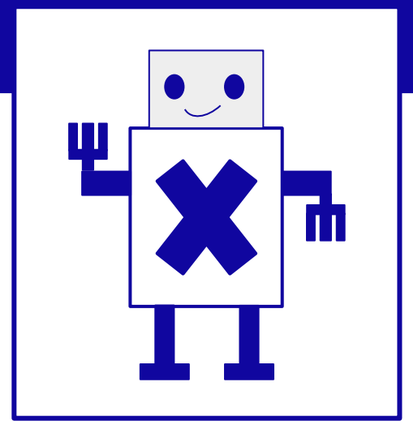
- **Lupa:** utilizando la cámara trasera, escala el espacio alrededor de usuario.

- **Los tamaños del texto dinámicos**: se puede cambiar el tamaño del texto en las apps que lo soportan.

- **Lectura de pantalla:** lee el contenido de pantalla.

- **Acceso guiado:** limita el acceso de usuario a solo una app.

- **AssistiveTouch:** ayuda a los usuarios que tienen dificultades para tocar la pantalla. Muestra un menú adicional en blanco y negro con los botones que permiten manipular el contenido de pantalla fácilmente.



Definición

Es un sistema de integración continua de Apple para los proyectos de iOS y Mac.

CARACTERÍSTICAS PRINCIPALES

Para usar el sistema se necesita instalar XCode Server. Está integrado en XCode a partir de la versión 9.

Es una **plataforma de integración continua que está integrada en XCode y que permite ejecutar los test unitarios**, los test de la interfaz gráfica, archivar los builds y mandar los correos electrónicos de estado.

Se encarga de crear y mantener los certificados y los perfiles de tu aplicación. También, muestra la cobertura de los test unitarios, ejecuta los test en diferentes dispositivos.

Además, **permite instalar los build *OTA*** *(Over-the-Air)* a través de navegador en tu dispositivo.

BLOQUES PRINCIPALES

- **Bot:** es un conjunto de tareas que quieres hacer con el código de repositorio.
- **XCode Server:** es una herramienta que permite ejecutar los Bots tanto localmente como en otro ordenador.
- **MacOS Server:** es una aplicación Mac que complementa MacOS con las funcionalidades de servidor de diferentes tipos como MTA servidor, AFP y SMB servidor, el servidor web, servidor wiki, servidor de mensajería, etc.
  Está disponible en App Store como la aplicación separada. Originalmente era necesario instalarlo pero actualmente ya no es necesario para poder utilizar los bots.

DESVENTAJAS

- Se puede usar solo para proyectos iOS/Mac.
- La funcionalidad está bastante limitada comparando con otras soluciones (Jenkins, Bamboo, etc).
- Tiene pequeños bugs.

VENTAJAS

- Se puede usar localmente en tu ordenador.
- Está integrado en XCode y no hace falta instalar nada más.
- Se ejecutan los test UI en diferentes dispositivos iOS/Mac.

¿Qué es?

Con App Clips **podemos crear un versión «reducida» de nuestra app que se ejecutará con un disparador como un código QR o una etiqueta NFC.** Están pensados para crear una experiencia limitada y contenida de lo que la app principal puede ofrecer, como vender algún producto o mostrar algún tipo de información.

VENTAJAS DE NEGOCIO

Los Clips son una **buena herramienta para aumentar la conversión de los potenciales usuarios** al no tener que buscar nuestra app en el App Store.
Los Clips pueden mostrarse con los siguientes disparadores:

- Escanear una etiqueta NFC o código visual en una ubicación física.
- Al tocar una sugerencia basada en la ubicación de Siri Suggestions.
- Tocando un enlace en la aplicación Mapas.
- Al tocar un banner de aplicación inteligente en un sitio web.
- Tocando un enlace que alguien ha compartido en la aplicación Mensajes.

LIMITACIONES

Los Clips presentan algunas limitaciones a la hora de desarrollarlos que se deben tener en cuenta:

- Disponibles con iOS 14.
- No pueden pesar más de 10 megas.
- Ha de ser nativo 100%.
- La construcción ha de ser 100% con SwiftUI.
- No puede acceder a los datos de usuario, por lo que los frameworks relacionados con éste no están disponibles (Salud, Contactos...).
- Al cabo de un tiempo de inactividad, el clip es borrado por el sistema.

DESARROLLO

Para añadir un App Clip a tu aplicación principal basta con crear un Target nuevo tal y como lo hacemos con las extensiones existentes, no hay limitación al número de clips que podemos añadir.

Los clips permiten programar una notificación tras 8 horas después de su descarga para recordar algo al usuario, al pulsarla, se vuelve a abrir el clip.

Hay una nueva API para verificar que el usuario se encuentra en el lugar donde debe estar el Clip.

Los permisos que demos al Clip los hereda la app principal en caso de descarga de ésta.

Definición

Con la ayuda de WidgetKit **puedes crear los widgets (mini-vistas) para tres plataformas de Apple: iOS, iPad OS y macOS**. A partir de iOS 14, puedes poner los widgets en el escritorio de tu iPhone, en TodayView en iPad OS y en el centro de notificaciones de macOS Big Sur (11.0).

CARACTERÍSTICAS PRINCIPALES

Los widgets son una manera dinámica y directa para demostrar la informacion útil de tu aplicación en el escritorio de tu móvil, tablet o portátil. El widget tiene que ser relevante, personalizable y entendible de un solo vistazo. La extensión de WidgetKit es una extensión de fondo (background), que proporciona las vistas del widget de antemano para cada momento específico. Así, se evita el problema de “carga” y “espera”. Los widgets se configuran a través de Intents. Además, se pueden actualizar las jerarquías de vistas ya preparadas desde tu aplicación o establecer un horario de actualización. Se puede crear una pila de widgets que va cambiando el widget que se muestra arriba de todos.

LIMITACIONES

- Está disponible solo a partir de iOS 14 y macOS Big Sur(11.0).
- Los widgets no pueden contener los elementos interactivos (como los campos de texto o los botones).
- Tienen solo tres tamaños (pequeño, medio y grande).
- Tienen que ser escritos con SwiftUI.
- Los widgets no tienen el estado.
- No soportan el scrolling.
- No soportan el reproducción de los videos o las imágenes animadas.
- La única interacción que está permitida es a través de tocar el widget que está vinculado a un enlace profundo (deepLink) de tu aplicacion o creando subenlace dentro del widget con Link API.

DESARROLLO

WidgetKit soporta SwiftUI, AppKit y las apps de macOS basadas en Catalyst. El widget se define a través de:

- **kind**: el identificador de tipo String de widget.
- **configuration:** el widget puede ser estático (*Static Configuration*) o configurable a través de Intent (IntentConfiguration).
- **supportFamilies**: el tamaño de widget: *systemSmall, systemMedium y systemLarge*.
- **placeholder**: la vista que muestra la estructura de widget sin mostrar los datos de usuario.
- **provider:** la estructura que proporciona el conjunto de vistas para cada momento de tiempo y la primera entrada de datos (*snapshot*).

Estas fichas son un recurso vivo,
¡anímate a hacernos sugerencias y comentarios!

@autentia

Puedes descargar la última versión en:

https://autentia.com/libros/

Además encontrarás libros gratuitos, guías y recursos útiles para tu día a día.